Mikko Aarnio

Insinöörityö

Tetris-verkkopelin toteutus Javalla

Metropolia Ammattikorkeakoulu
Insinöörityö
Tietotekniikka
Tetris-verkkopelin toteutus Javalla
26.11.2010

Tiivistelmä

Metropolia

Tekijä Otsikko Sivumäärä Aika	Mikko Aarnio Tetris-verkkopelin toteutus Javalla 45 sivua 26.11.2010
Tutkinto	Insinööri
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	lehtori Miikka Mäki-Uuro lehtori Jorma Räty

Tämä insinöörityö käsittelee Tetrikseen pohjautuvan verkkopelin toteuttamista Java-ohjelmointikielellä. Työ tulee käsittelemään MVC-mallin mukaista hierarkian rakentamista, yhteyden hallitsemista Socket-TCP-rajapinnalla ja todennäköisten ongelmatilanteiden ratkaisemista. Työ opastaa graafisten työkalujen käyttämisessä ja näppäinten kuuntelun toteuttamisessa, käyttäen koodiesimerkkejä.

Työssä tutustutaan ikkunoiden sekä painikkeiden tekoon Swing-komponenteilla sekä opetetaan käyttämään ActionListener- sekä KeyListener-kuuntelijoita syötteiden lukemiseen. Työ perehdyttää myös käyttämään Graphics2D-työkalua ja sisältää myös hyödyllisiä vinkkejä samankaltaisten pelien tekemiseen.

Avainsanat	Java, Socket, Graphics2D, ActionListener, KeyListener, Tetris

Abstract

Metropolia

Author(s) Title Number of Pages Date	First name Last name Tetris network game with Java 45 pages 26 Nov 2010
Degree	Engineer
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Miikka Mäki-Uuro, Senior Lecturer Jorma Räty, Senior Lecturer

This thesis describes programming a Tetris-like online game with Java as the programming language. The work unfolds using MVC-model hierarchy, Socket-TCP connection control, solving of possible problems, using of drawing elements and listening keyboard inputs with the help of code examples.

The reader of this thesis will learn about window and button creation with Swing-components and will be taught to use ActionListener and KeyListener as input readers. This work also introduces the Graphics2D tool and gives valuable tips for creating same kind of games.

Keywords	Java, Socket, Graphics2D, ActionListener, KeyListener, Tetris

SISÄLLYS

1 JOHDANTO

Opinnäytetyössä tehdään verkkopeli Tetriksestä. Työssä kuvataan ohjelman rakennetta ja arkkitehtuurin toteuttamista. Ohjelman koodauksessa on käytetty Internetistä löydettyä materiaalia.

Opinnäytetyön tavoitteena on tehdä oma Tetrikseen pohjautuva moninpeli-versio, jossa toteutetaan yksinpeli ja verkon välityksellä tapahtuva moninpeli kahden koneen välillä. Työn tarkoituksena on myös toimia oppaana henkilöille, jotka ovat kiinnostuneet ohjelmoimaan pelejä tai muita graafisia sovelluksia Javalla. Työssä selvitetään myös todennäköisiä ongelmakohtia moninpelin ohjelmoimisessa ja laaditaan ohjeistus ongelmakohtien selvittämiseen.

Opinnäytetyö käsittelee verkkopelaamisen toteuttamista ohjelmoijan näkökulmasta. Työ rajataan Javan Swing-luokkahierarkiaan ja Socket-rajapintaan. Työssä ei käsitellä 3-ulotteista piirtämistä. Ohjelman hierarkia kuvataan pääpiirteittäin käyttäen UML-kaavioita.

Aloitin tekemään kyseistä opinnäytetyötä siitä syystä, että olen kiinnostunut peliohjelmoinnista. Pelejä olen pelannut pikkupojasta lähtien, ja ne ovat minun intohimoni. Olen myös aloittanut tekemään pelejä harrastuspohjalta ja olen tähän mennessä toteuttanut noin 10 peliä pokerin peluusta shakkiin, sekä noin 5 muuta graafista kokeilua, joihin kuuluu mm. painovoimasimulaattori. Toivon työni antavan lukijoille paljon hyödyllistä tietoa uusien välineiden parissa joita toivon heidän hyödyntävän parhaansa mukaan.

Työssä käydään läpi Tetriksen historiaa ja suomalaisille tutuksi tulleita pelin variaatiota. Työssä perehdytään toteutetun Tetriksen teossa käytettyihin Java-ohjelmiston termistöihin. Työssä käsitellään tiedonsiirtoa UDP- ja TCP-protokollilla sekä käydään läpi ohjelman ja moninpelin toteutusta. Li-

säksi pohditaan yhteyden muodostamiseen, ylläpitämiseen ja katkaisemiseen liittyviä ongelmatilanteita.

2 VERKKOPELAAMISEN JA TETRIKSEN HISTORIAA

Tässä luvussa käydään läpi työn tavoitteena olleen ohjelman taustaa. Aluksi esitetään verkkopelaamisen historiaa ja käydään läpi, miten yhdeltä koneelta pelatut moninpelit muuntuivat ajan saatossa verkkopeleiksi. Tämän jälkeen esitetään Tetriksen historiaa. Lisäksi paneudutaan Tetriksen mekaniikkaan ja sitä seuranneisiin variaatioihin ja niiden toimintaan.

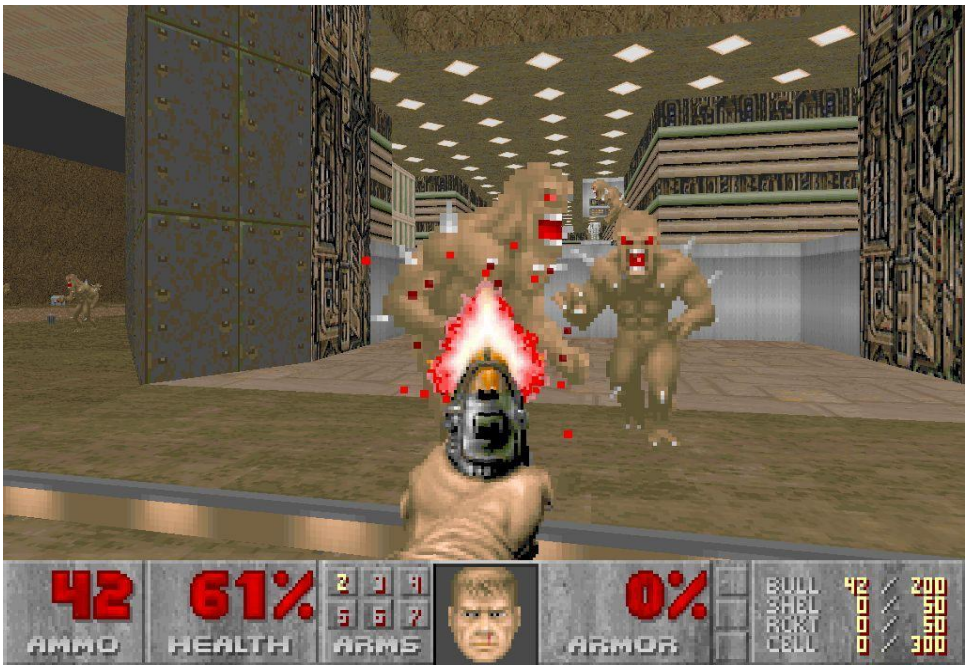
2.1 Verkkopelaamisen historia

Verkkopelit tai toisinsanottuna nettipelit ovat yksi tietokone- ja konsolipelaamisen muoto, jossa pelaajat ottavat mittaa toisistaan tai pelaavat yhdessä tietokonevastustajia vastaan verkon välityksellä. Pelaaminen ihmispelaajia vastaan on luonnollisesti hauskempaa johtuen pelaajien ennustamattomuudesta.

Jopa ensimmäiset konsolipelit omasivat kahden pelaajan välisen kommunikaation, mutta tämä tapahtui aina saman laitteen alla jolla peliä pelattiin. Järkevän tekoälyn tekeminen oli silloin suorittimien hitauden vuoksi hankalaa. Hyvänä esimerkkinä moninpelin suosiosta on vuonna 1972 Atarin julkaisema Pong, jossa pelaajat ottavat mittaa kaksiulotteisessa tennismatsissa. Pong oli suuri hitti, ja kymmenet yhtiöt valmistivat samankaltaisia Pong-variaatioita omille televisioon kytkettäville konsoleilleen. [1.]

Internetiä alettiin perustaa 1960-luvulla Yhdysvaltojen puolustusministeriön toimesta, jolloin tarkoituksena oli luoda hajautettu viestintäjärjestelmä, jolla pyrittiin vihollisen täsmäiskuja kestävään viestintäkoneistoon [2.]. Internet alkoi saapua kuluttajamarkkinoille vasta 1990-luvun alussa laajan verkottautumisen myötä ja ensimmäiset verkossa pelattavat moninpelit

alkoivat tehdä tuloaan. Vuonna 1993 julkaistiin vallankumouksellinen, moninpelattava tietokonepeli Doom (kuva 1). Pelissä pelaaja omaksuu ihmishahmon ja pelaa tämän katseen sisältä kolmiulotteisessa maailmassa. Moninpeliominaisuuksissa voidaan valita Co-op jossa päästään pelaamaan maksimissaan neljän pelaajan kanssa yksinpelikenttiä kampanjamoodissa. Suuren suosion saavutti kuitenkin Deathmatch, jossa tavoitteena oli hävittää mahdollisimman monta pelaajaa tietyn ajan sisällä.



Kuva 1. Doom oli alansa uranuurtaja niin graafisella puolella kuin moninpeli ominaisuuksilla. [3.]

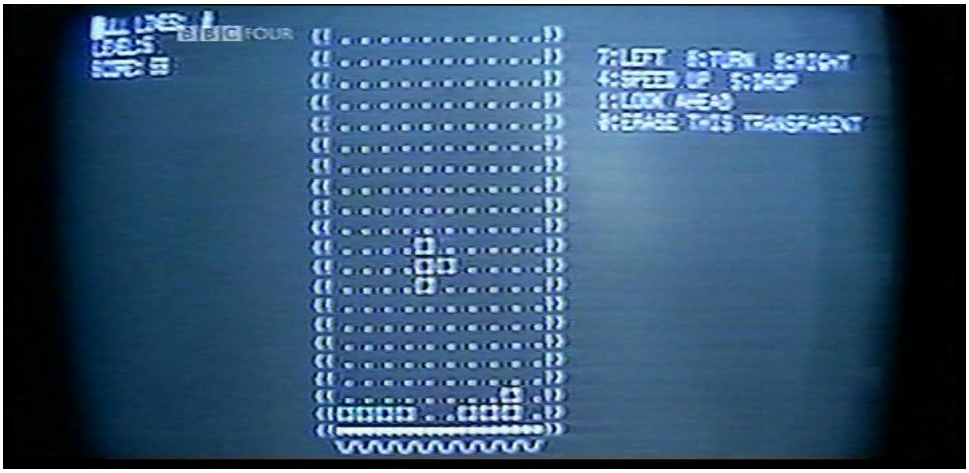
1990-luvun lopulla monet peliyhtiöt alkoivat keskittyä moninpelaamiseen. Ensimmäinen yli sadantuhannen pelaajan Massive Multiplayer Online Role Playing Game eli MMORPG saapui vuonna 1997 kantaen nimeä Ultima Online[4.]. MMO tarkoittaa, että pelaajat liittyvät yhteiseen peliin joissa voi yhtä aikaa olla sisällä tuhansia pelaajia. MMORPG on peli, jossa ei ole varsinaisia aloitus- tai lopetustapahtumia vaan peleihin liitytään kuluttamaan aikaa muiden pelaajien parissa. Tällä hetkellä MMORPG-maailman huippua pitää Blizzard-yhtiön vuonna 2004 julkaisema World of Warcraft (kuva 2). Peli on saavuttanut jo yli 12 miljoonan pelaajan rajan, eikä pelaajien määrän kasvulla näy loppua[5.].

Kuva 2. World of Warcraft ja 40 pelaajan luolasto. [6.]

Kuten edellä näemme, on moninpeli yleistymässä hurjaa vauhtia. Saattaa hyvin olla, että 50 – 100 vuoden päästä ei valmisteta muita kuin moninpelejä.

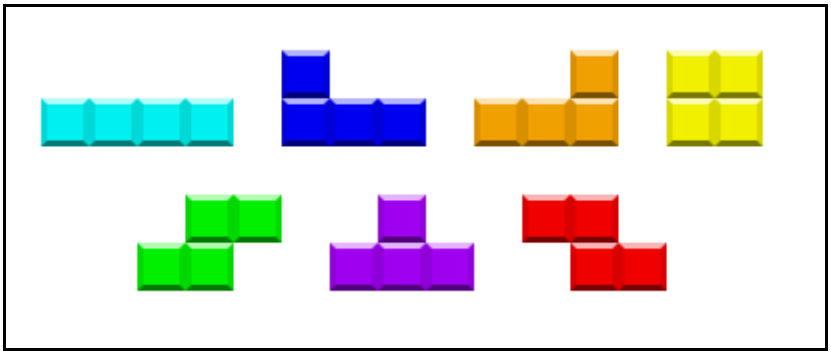
2.2 Tetriksen historia

Tetris on tietokonepeli, jonka on kehittänyt neuvostoliittolainen pelien suunnittelija Aleksei Pažitnov vuonna 1984 Elektronika 60 -konsolille (kuva 3). Vuonna 1986 Tetris saavutti huikean suosion saavuttuaan kotitietokoneille. Suomessa peli arvosteltiin vuonna 1988 saaden MikroBittilehden arvostelussa täydet 5 tähteä.



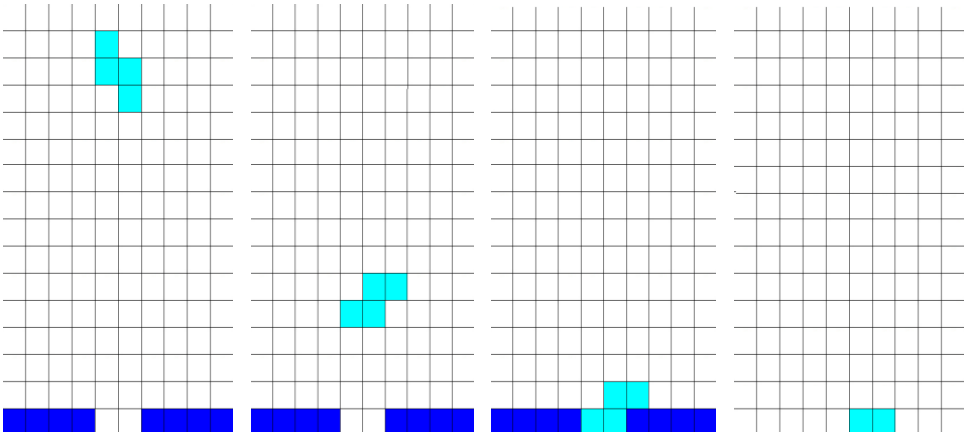
Kuva 3. Tetris Elektronika 60 konsolilla. [8.]

Alkuperäinen Tetris on yksinpelattava puzzle-peli, jonka ideana on toimia palikoiden kasaajana. Palikoita on 7 erilaista (kuva 4).



Kuva 4. Tetriksestä löytyy 7 erilaista 4:stä ruudusta koostuvaa palikkaa.

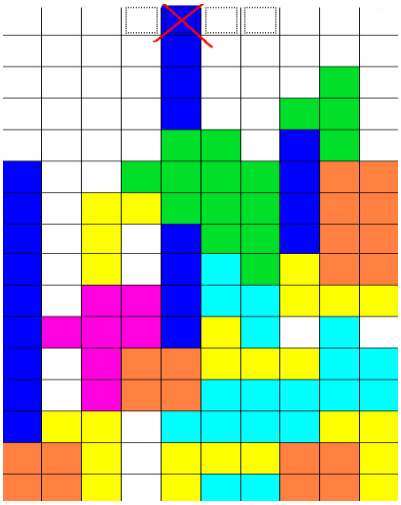
Peli pelataan ruudukossa, joka on 10 ruutua leveä ja noin kaksi kertaa korkea (kuva 5). Käynnistyessä ruudukko on tyhjä, ja siihen ilmestyy satunnaisessa järjestyksessä neljästä ruudusta koostuvia palikoita, joita pelaajan on tarkoitus pudottaa ruudun alareunaan muodostaen näillä täysiä vaakarivejä.



Kuva 5. Tetriksen eteneminen.

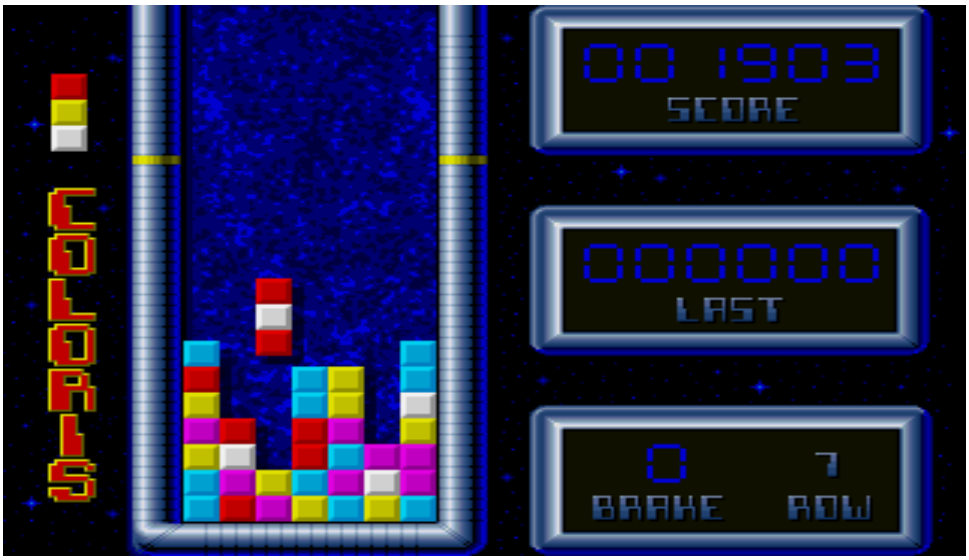
Palikat ilmestyvät ruudun yläreunaan keskelle, josta ne lähtevät liikkumaan kohti alareunaa tasaisella nopeudella. Palikkaa voidaan käännellä 90 asteen kulmassa 360 astetta. Kun palikka ei voi enää liikkua alemmas, se pysähtyy. Pysähtyneistä palikoista on tarkoitus muodostaa täysi rivi, jolla palikat saadaan katoamaan ja ylemmät rivit siirtyvät yhden rivin alaspäin.

Kun pelaajan onnistuu syystä tai toisesta jättää jokaiselle vaakariville yksi tai useampi palikkaväli, jonka seurauksena pino kasvaa, on vaarana se, että peli loppuu. Tämä tapahtuu kun palikat eivät mahdu enää ilmestymään ruudukkoon (kuva 6).



Kuva 6. Peli päättyy, kun seuraava palikka ei mahdu ilmestymään.

Peli sai suuren suosionsa vuoksi paljon seuraajia, jotka matkivat sen fysiikkaa. Coloris on suomalainen Amigalle tehty Tetris variaatio, joka julkaistiin vuonna 1990 (kuva 7). Sen tavoite on sama, eli kerätä pisteitä ja estää palikkapinoa karkaamasta liian korkeaksi. Palikat putoilevat Tetriksen tavoin ylhäältä alaspäin mutta sillä erolla, että palikat ovat vain kolmen ruudun mittaisia pystypalikoita eikä niiden kulmaa voida vaihtaa. Keskeisenä teemana toimivat värit, joita on tarkoitus saada kolme vierekkäin. Tällä mekanismilla katoavien palikoiden yläpuolelta tippuvat palikat voivat saada aikaan sarjapudotuksia, joissa useita palikoita häviää kerralla.



Kuva 7. Suomalainen Coloris oli suomen vastine Tetrikselle. [9.]

Vuonna 1999 suomalainen peliyhtiö Ninai Games julkaisi tietokonepelin Drop Mania (kuva 8). Tässä pelissä tarkoituksena on tuttuun tyyliin estää oman pinon kasvamista yläreunaan asti sekä läpäistä yksinpelissä tasoja keräämällä tarpeeksi pisteitä tyhjennetyillä palikoilla. Osassa yksinpelitehtäviä joutuu kamppailemaan alhaalta nousevaa palikkajonoa vastaan, toisissa taas on tarkoituksena tyhjentää kenttä. Tämä tyhjentäminen tapahtuu Tetriksestä ja Coloriksesta poiketen harvinaisilla räjäytyspalikoilla, joita tulee harvoin. Palikoita pudotetaan kaksi kerrallaan. Palikat tulevat automaattisesti ruudun yläosassa sijaitsevaan pudotuspalkkiin, jossa valitsimella saattaa vaihtaa kahden viereisen nappulan paikkaa. Kun nappulat on pudotettu, uudet nappulat ilmestyvät vanhojen tilalle. Samanväriset nappulat liimautuvat yhteen ja muodostavat uusia kokonaisuuksia. Kun pudotettu räjähdyspalikka on kosketuksissa saman värin palikan kanssa, molemmat katoavat. Tällä keinolla saadaan myös monta kombinaatiota, kun räjäytyspalikat eivät pudottaessa ole heti kosketuksissa samanväristen kanssa ja myöhemmin välikappaleet, jotka estävät kosketuksen katoavat. Pelin jatko-osa, Super Drop Mania, julkaistiin Nokian Symbian- sekä Windows Mobile-laitteille vuonna 2005.

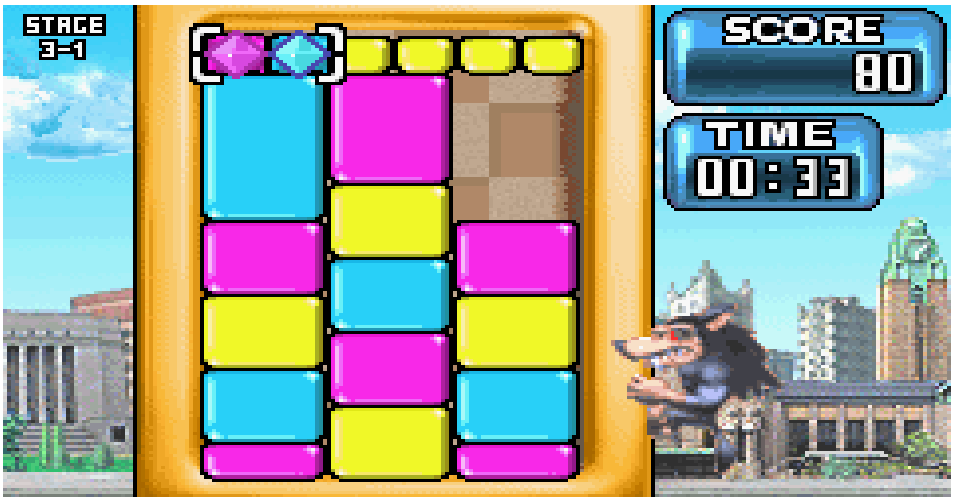
Kuva 8. Super Drop Mania kännykällä. [10.]

Vuotta myöhemmin, vuonna 2000, Ninai Games julkaisi pelin nimeltä Puzzle Station (kuva 9). Se jatkaa Drop Manian viitoittamaa tietä eli pudottaa kaksi palikkaa kerralla. Nyt erikoispudotusnappulat on poistettu, ja nappuloita hävitetään, mikäli niitä on 4 vierekkäin. Hopeanväriset palikat eivät tuhoudu, ja ne saa jättää jäljelle. Peli on keskittynyt enemmän ongelman ratkontaan ja tehtävänä on tyhjentää pelilauta kaikesta värillisestä. Kenttiä on jopa viisikymmentä.

Kuva 9. Puzzle Station maksoi kotijäätelössä pari euroa. [11.]

Ninai Games julkaisi myös vuonna 2001 kannettavalle käsikonsolille GameBoy Advancelle pelin Rampage Puzzle Attack (kuva 10), joka oli myös

ensimmäinen suomalainen konsolipeli. Pelin mekaniikka on lähes sama kuin Drop Maniassa, mutta se on saanut vaikutteita vuotta aikaisemmin julkaistusta Puzzle Station-pelistä. Pelissä on monta erilaista moodia, joissa yhdessä on tehtävä Drop Manian tavoin tyhjentää kenttä. Toisessa on tarkoitus päästä pelastamaan pelikentän pohjalla häkissä oleva eläin ennen ajan päättymistä. Esteenä toimivat luonnollisesti väripalikoista muodostuva kerros. Eräs moodi keskittyy ongelman ratkontaan, jossa vain rajallisella määrällä nappuloita pitää selvitä kentästä läpi.



Kuva 10. Rampage Puzzle Attack vuodelta 2001. [12.]

Tetriksestä on tullut lähes rajaton määrä erilaisia variaatioita, ja se on avannut sitä kautta uusia ideoita pelinkehittäjille. Tetriksen suosio oli ja on edelleen valtava. Jopa uusimmille puhelimille sekä multimedialaitteille tulee yhä uusia Tetris-versioita.

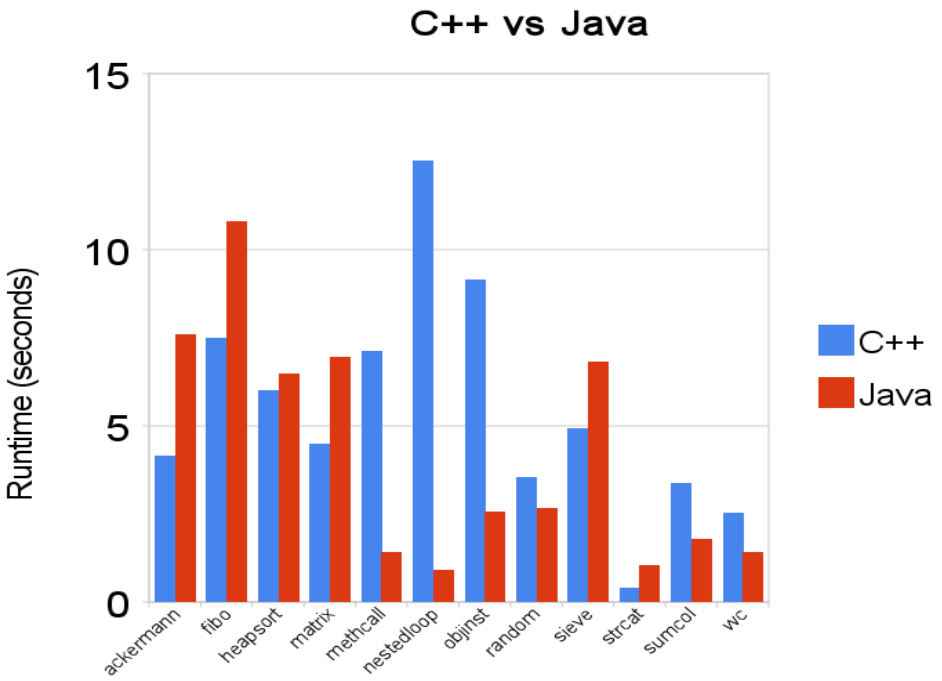
3 TOTEUTUSTEKNOLOGIAT

Seuraavaksi käydään läpi Java pääpiirteisesti sekä työssä käytetyt Swing-kirjaston komponentit. Tämän jälkeen kerrotaan MVC-mallista ja tämän hyödyntämisestä ohjelmoinnissa.

3.1 Javasta yleisesti

Ohjelma on toteutettu Java-ohjelmointikielellä, joka kehitettiin vuonna 1995 ja on nykyään suosituimpia ohjelmointikieliä. Java mielletään yhdeksi helpoimmista ohjelmointikielistä, sillä siinä on muun muassa automaattinen roskien kerääminen. Tämä tarkoittaa sitä, että ohjelma poistaa muistinvarauksen oliolta, mikäli se on hukattu, eikä sitä voi enää käyttää.

Java on ollut aikaisemmin melko hidas ohjelmointikieli, mutta tämä on muuttunut viimeisen kymmenen vuoden aikana rajusti. Nykyään Java on C++:n kanssa tasoissa ja menee ohi monessa vertailussa (kuva 11). Tämä on hyvin tärkeää, sillä pelit pyrkivät olemaan visuaalisesti miellyttäviä. Tämä tarkoittaa nopeaa laskentatehoa ennen ruudunpäivitystä tapahtuvassa laskennassa sekä piirtämisessä.



Kuva 11. Vertailu Javan ja C++:n välillä, vuosi 2010. [13.]

Java-pelin tekemiseen tuskin vaaditaan erillisiä kirjastoja, sillä sen kirjasto sisältää monia välineitä graafisestikin haastavien ohjelmien tekemisiin.

3.2 Swing

Swing on tässä työssä käytetty Javan graafisten käyttöliittymien piirtämiseen erikoistunut luokkakirjasto, josta löytyvät kaikki tarvittavat piirtoon tarvittavat komponentit ohjelmien tekoon. Sen kehitti vuonna 1996 Netscape. Kehityksen tarkoitus oli tuottaa helposti vähäistä muokkaamista vaativa luokkakirjasto graafisten ohjelmien piirtoon. Kirjasto on ollut Javan vakiotyökalu 1.2:sta lähtien. Näitä Swingin elementtejä kutsutaan myös J-komponenteiksi. Seuraavaksi käydään lyhyesti läpi työssä käytetyt J-komponentit

3.2.1 JFrame

JFrame on graafisen käyttöliittymän pohja, jolle ohjelman piirto rakentuu. JFramessa on mahdollisuus syöttää ikoni sekä nimi, jotka näkyvät sinisessä yläpalkissa. JFrame sisältää myös pienennys, koon kasvatus sekä sulkemisnappulat, jotka voi halutessaan ottaa pois päältä tai määrätä omat toiminnot nappien painalluksille. JFrame, tai mikä tahansa muu J-komponentti voidaan luoda muuttujaksi, tai se voidaan periä luokkaan.

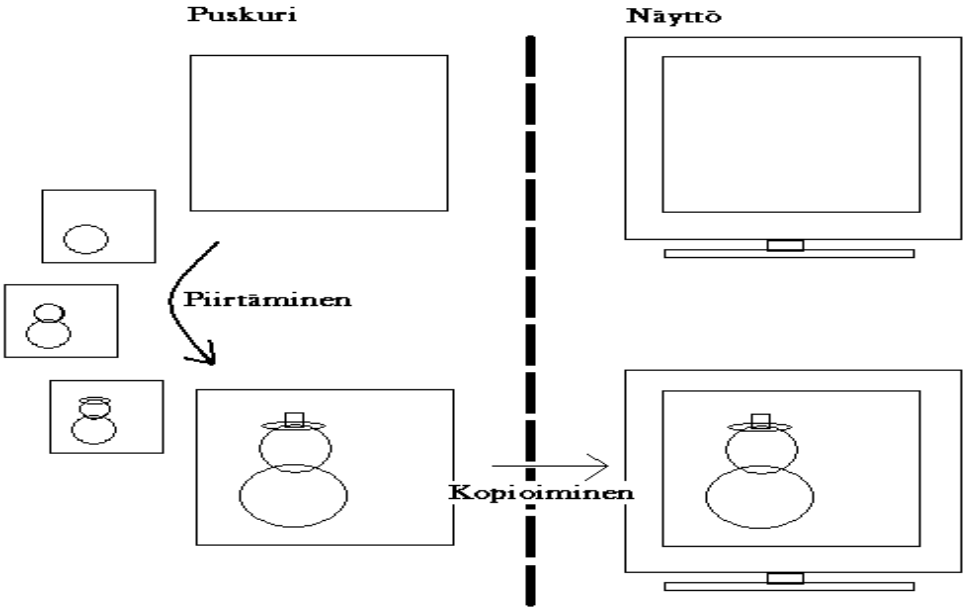
3.2.2 JMenuBar, JMenu, JMenuItem

JMenuBar on ohjelman ylälaitaan sijoitettava valikkosäiliö, johon lisäämällä JMenu-säiliöitä rakentuu vasemmalta oikealle valikkonäppäimiä, jotka avaavat niihin lisättyjä valintoja. JMenu on valikkonäppäin joka sisältää valikkopainikkeita. Valikko painikkeet ovat JMenuItem komponentteja joihin usein liitetään ActionListener-tapahtumakuuntelijoita. JMenuItemeihin voidaan myös liittää luokan omia näppäinyhdistelmiä kuuntelevia komponentteja.

3.2.3 JPanel

JPanel on alemman tason säiliö, johon voidaan liittää muita alemman tason komponentteja, mukaan lukien toisia JPaneleita tai kohdistaa piirtämiskomentoja. Piirtäminen tapahtuu joko `paint` tai `paintComponent` komennolla, saaden parametrina Graphics-piirtokirjaston. JPaneeleihin voidaan tuottaa reunukset `setBorder`-metodilla.

JPanel käyttää kuvan puskurointia automaattisesti näytön vilkkumisen estämiseksi, jolloin kuva tuotetaan valmiiksi puskurissa ennen kuin se piirretään näytölle sen sijaan, että jokainen piirto tehtäisiin yksi kerrallaan (kuva 12).



Kuva 12. Puskurin toiminta.

Ohjelma kirjoittaa puskuriin(offscreen) kuvan valmiiksi, jonka jälkeen tämä kopioi kuvan ruudulle. Halutessaan tuplapuskuroinnin saa pois päältä käyttämällä `setDoubleBuffered`-metodia ja asettamalla parametriksi false. Tällä saatetaan parantaa ohjelman suorituskykyä mutta se vähentää käyttäjäkokemusta. Kyseinen offscreen-tekniikka voidaan myös tehdä manuaalisesti kirjoittamalla erillinen Graphics-offscreen kuva, jota sitten `paint` tai `paintComponent`-metodissa kutsutaan.

Graphics on graafinen työkalu jolla voidaan piirtää kuvia sekä monimutkaisia kuvioita. Graphics on vanhentunutta tekniikkaa, mutta se pitää sisällään uudemman kirjaston 2D-piirtoon, jonka käyttöönotto tapahtuu uuden Graphics2D-muuttujan luonnilla ja Graphics-työkalun uudelleen nimeämisellä, mikäli monimutkaisempia kuvioita tarvitsee työstää.

Graphics2D-kirjastolla voidaan Graphics-kirjastosta poiketen esimerkiksi kääntää piirrettävää alustaa siten, että piirto jatkuu alustan mukaisesti mutta alusta näkyy vinossa. Graphics2D:llä voidaan myös skaalata kuvaa, joten vaikka ohjelma olisikin toteutettu vain tietyn kokoiselle ikkunalle, voi skaalaamista käyttäen vetyttää kuvan kokoa halutun suuruiseksi.

3.2.4 JButton

JButton on ruudulle piirtyvä nappula jonka tarkoitus on kuunnella hiiren painalluksia rajojensa sisällä. Tapa painallusten kuunteluun on ActionListener-luokka.

3.2.5 JTextField ja JLabel

JTextField on tekstin syöttämiseen tarkoitettu kenttä, mutta siihen voidaan myös valmiiksi syöttää haluttua dataa. JLabel on nimensä mukaan kyltti, jonka tarkoitus on antaa nimi vieressä oleville syöttökentille.

3.3 MVC-malli

Pelin arkkitehtuuri perustuu Model-View-Controller malliin(lyhennetään MVC). Ajatuksena on, että ohjelma toteutetaan irtonaisesti niin, että sen osaa muuttamalla ei tarvitse muuttaa muita osia sopeutuviksi. MVC-mallia käytetään erityisesti graafisesti haastavien ohjelmien suunnittelussa, joka tekee esimerkiksi tuotteen muuntamisen pelikonsolilta toiselle vaivattomammaksi.

Model tarkoittaa mallia, eli ohjelman tiedon laskenta- ja käsittelyelintä. Model ei koskaan toimi käyttäjän toimintojen kuuntelijana eikä se myöskään käsittele näytölle piirtämistä.

View on vastuussa näkymästä, eli kaikesta ohjelman ruudulle piirtämästä materiaalista. View ei näe muita luokkia, vaan sille annetaan muista luokista ohjeet.

Controller on ohjelmaa hallitseva osa, jolla käyttäjä vain pystyy muuttamaan ohjelman normaalia kulkua. Controller kuuntelee käyttäjän tapahtumia ja aiheuttaa sen pohjalta muutoksia muihin luokkiin.

3.4 ActionListener

ActionListener on komponentteihin kohdistuviin tilan muutoksiin reagoiva kuuntelija. Helpoin tapa toteuttaa ActionListener on sopia kuunneltaville komponenteille ActionCommand eli toimintokomento, jolla kuuntelija tunnistaa painalluksen ja osaa sovittaa sen oikeaan ohjelmoijan tekemään metodiin. Kun kuunneltavaa kenttää painetaan, niin käynnistyy `actionPerformed`-metodi. Se saa parametrikseen ActionEvent-muuttujan, josta saatavaa actionCommand-arvoa voidaan verrata aikaisemmin sovittuun komentoon. Toinen vaihtoehto olisi tarkistaa, onko kyseisen komennon lähde sama kuin painikkeen tyyppi, joka on minusta hitaampi ja hankalampi toimenpide, koska se vaatisi kyseisten kappaleiden tuntemista kyseisessä luokassa, tai niiden erillistä linkitystä alaluokilta. Itse suosittelisin käyttämään tätä, jos moni painike omaa samankaltaisen metodin, kuten esimerkiksi miinaharavassa, jossa painikkeita vähintään sata.

3.5 KeyListener

KeyListener on käyttäjän näppäimistön painalluksiin reagoiva kuuntelija. Jokainen näppäinpainallus kuuntelijan kohdistuksen alla aiheuttaa `keyPressed` kutsun. `keyReleased` käynnistyy painalluksen jälkeen. `keyTyped` toimii samalla periaatteella kuin `keyPressed`, mutta se tallentaa vain näppäimistön merkkien painallukset ja ne näkyvät vain `getKeyChar`-metodilla.

3.6 Output- ja InputStream

OutputStream tarkoittaa ohjelman tietovirtaa, jossa tieto siirretään ohjelmasta ulospäin. Tätä käytetään tallentamisessa. InputStream tarkoittaa tietovirran tuomista sisäänpäin. Tätä käytetään tiedoston lataamisessa.

3.7 Serializable

Serializable on rajapinta, jossa ei ole yhtään metodia. Serializable mahdollistaa rajapinnan omaavan luokan muuntamisen objektiin. Rajapinta on tärkeä objektien tallentamisessa, sillä tallentamisvälineissä voidaan käyttää objekteja tiedostoon vievää ObjectOutputStream-luokkaa.

3.8 Thread

Thread on suomeksi säie. Säikeiden avulla voidaan ohjelmassa ajaa rinnakkain useampaa tehtävää. Javassa säikeitä voidaan luoda kahdella tavalla; Joko periyttämällä oma luokka luokasta Thread tai implementoimalla Runnable-rajapinta. Tällä voidaan mahdollistaa operaatio, kuten palikan pudottaminen puolen sekunnin välein sekä samanaikaisesti palikan siirtäminen oikealle näppäinkomennolla. Ohjelmassa toimii aina vähintään yksi pääsäie, joka käynnistyy `main`-kutsulla.

3.9 Tiedon välitys

Tiedonsiirtotapoina käytetään (User Datagram Protocol)UDP- tai (Transmission Control Protocol)TCP-mallinnusta. Tässä pelissä on päätetty käyttää TCP:tä, koska osa lähetettävän paketin sisällöstä sisältää vain kerralla lähetettävää tietoa, sekä Client-Server mallia, joka on peleissä tyypillinen keino tiedon välitykseen. Tieto välitetään Javassa pistokkeiden avulla. Seuraavaksi tutustutaan UDP- ja TCP-malleihin sekä Client-Server-arkkitehtuuriin.

3.9.1 UDP

UDP eli User Datagram Protocol on nopeaan tiedon välittämiseen perustuva tiedonsiirtomenetelmä, jonka tavoite on saada tiedostot äkkiä perille, ikään kuin nopeutetulle liukuhihnalle, jolle pitäisi kasata paketteja nopeasti toisen perään, kiinnittämättä huomiota siihen, tulevatko paketit perille. UDP on hyvin suosittu peleissä, joissa pienikin viive on hyvin häiritsevä tekijä, kuten nopeatempoisissa FPS-peleissä ja joissa ei ole niin tärkeässä roolissa, jos jokin liikahdus jää tulematta perille. Toisaalta, jos asioidaan

esimerkiksi verkkopankissa, tulisi jokaisen tehdyn tiedonsiirron aina päätyä perille.

3.9.2 TCP

TCP eli Transmission Control Protocol on toinen tiedonsiirto protokolla, joka pyrkii valvomaan yhteydessä olevan koneen kanssa lähetettyjen pakettien perille menosta.

Jos halutaan varma keino saada tietoa perille, käytetään TCP-protokollaa, jossa palvelin ja asiakas pitävät kirjaa saapuvista paketeista ja niiden järjestyksistä. Mikäli jokin paketti on kadonnut matkalla, lähettää asiakas paketin uudelleen. Tämänkaltainen tiedonsiirtomenetelmä sopii erinomaisesti hidastempoisiin vuoropohjaisiin peleihin kuten shakki tai pokeri. Toisaalta, tästä tarkistelusta ja uudelleenlähettämisestä koituu haittaa, mikäli tieto on vanhentunutta, eikä sen tuonnissa ole järkeä. Tämänkaltainen tilanne voi syntyä esimerkiksi aikaisemmin mainitussa FPS-pelissä, jossa pelaaja on vaihtanut hahmon paikkaa. Silloin vanhan sijainnin lähettäminen vastapuolen pelaajalle aiheuttaisi vain harhaan ampumista.

TCP on yhteydellinen, eli siinä asiakas täytyy tunnistaa ja asettaa omaan pistokkeeseen. UDP-rajapinnassa tätä ei tarvitsisi toteuttaa, sillä UDP on yhteydetön. Tällä tunnistamisella voidaan myös kätevästi varmistaa, että asiakas on valmis aloittamaan pelin ilman, että tarvitsisi harrastaa sen kummempaa kikkailua.

Javassa TCP-yhteys muodostetaan ServerSocket-luokalla, joka jää `accept`-komennolla odottamaan liittyvää yhteyttä. Asiakas käyttää Socket-luokkaa yhdistääkseen. ServerSocket avaa yhteyden omaan Socket-luokkaan, mikäli yhteys onnistuu. Yhteys muodostetaan TCP/IP-osoitteiden välille. Käytetty portti voi olla mikä tahansa 0 - 65535 väliltä.

3.9.3 Client-Server

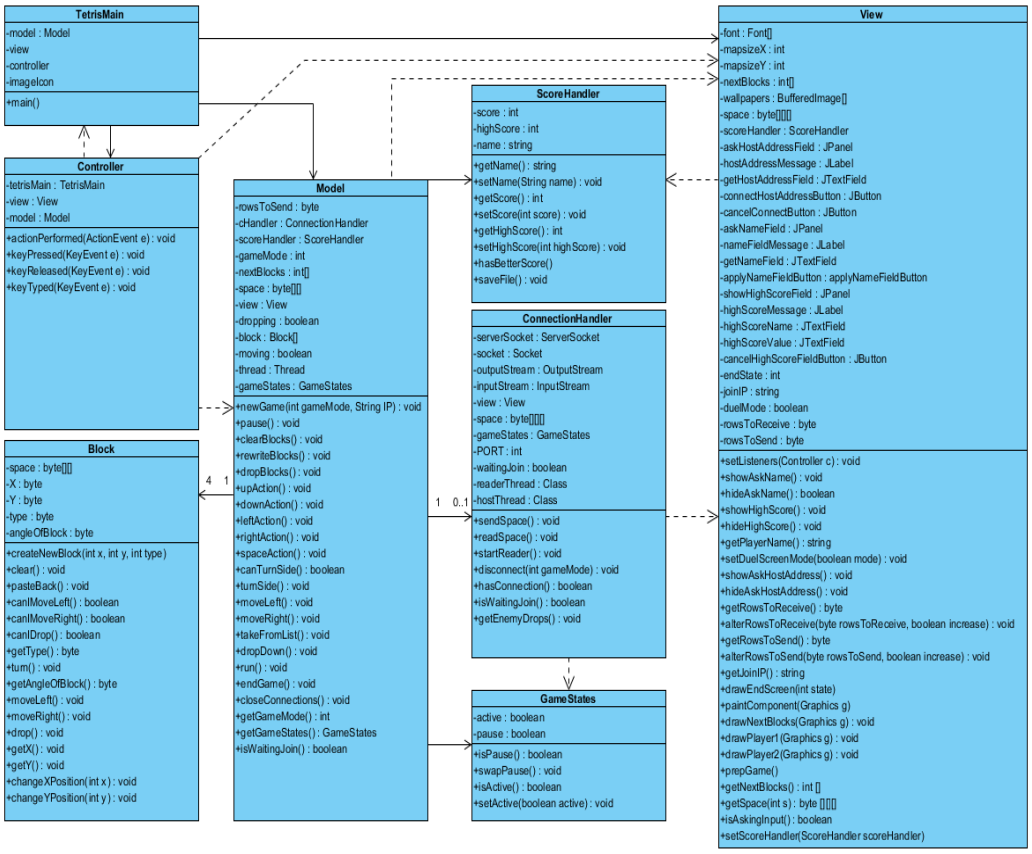
Client-Server on malli, jossa yhteyttä usean koneen välillä pidetään yllä keskittämällä yhteydet yhden koneen taakse. Tätä konetta kutsutaan palvelimeksi. Palvelimeen liitetyt koneet ovat asiakkaita ja ne ovat yhteydessä keskenään vain palvelimen kautta. Malli on hyödyllinen siitä, että se pyrkii minimoimaan kulkevan tiedon määrää. Tällä menetelmällä voidaan myös ulkoistaa osa laskemisesta palvelimelle. Toisenlaisia vaihtoehtoja olisi mm. Peer to Peer, tai lyhyemmin P2P-malli, jossa perustetaan vertaisverkko koneiden välille. Vertaisverkossa jokainen kone toimii sekä palvelimena että asiakkaana ja on suorassa yhteydessä kaikkien koneiden kesken.

4 TOTEUTUKSEN ARKKITEHTUURI

Tässä luvussa käydään läpi ohjelman rakennetta käyttäen apuna luokkakaaviota. Luvussa käydään läpi luokat yksi kerrallaan kertoen lukijalle pelin käynnistämisen, tiedon käsittelyn, piirtämisen, kuuntelemisen, tiedon lähettämisen sekä tallentamisen keskeiset toiminnot.

4.1 Yleisrakenne

Luokkakaaviossa on kuvaus ohjelmasta (kuva 13), jossa TetrisMain toimii käynnistävänä luokkana. TetrisMain luo Model, View sekä Controller luokkien ilmentymät.



Kuva 13. Luokkakaavio.

4.2 TetrisMain

TetrisMain toimii ohjelman käynnistävänä luokkana sekä ohjelman kehyksenä, johon muut piirtoelementit liitetään. Tässä osassa käydään luokan toiminta.

4.2.1 Ohjelman käynnistäminen

Ohjelman käynnistyessä Java avaa TetrisMain luokassa sijaitsevan `main`-metodin.

```
public static void main(String []args)
{
    new TetrisMain();
}
```

Kyseisessä metodissa luodaan uusi TetrisMain luokan ilmentymä.

4.2.2 TetrisMain-olion luonti

Pelin käynnistyessä luodaan TetrisMain-olio. Olion konstruktorissa luodaan Model-, Viev- sekä Controller-olio ja tuotetaan kehykseen ylävalikko sekä valikon painikkeet.

```
public class TetrisMain extends JFrame{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private Model model;
    private View view;
    private Controller controller;

    public TetrisMain(){
        view = new View();
        model = new Model(view);
        controller = new Controller(this,view,model);
        view.setListeners(controller);
        JMenuBar jmenubar = new JMenuBar();
        JMenu gameMenu = new JMenu("Game");
        JMenu otherMenu = new JMenu("Other");
        JMenuItem singleplayer = new JMenuItem("Start Singleplayer");
        JMenuItem multiplayerHost = new JMenuItem("Host Multiplayer");
        JMenuItem multiplayerJoin = new JMenuItem("Join Multiplayer");
        JMenuItem pause = new JMenuItem("Pause");
        JMenuItem highscore = new JMenuItem("Highscore");
        JMenuItem exit = new JMenuItem("Exit");
        JMenuItem copyIP = new JMenuItem("Copy IP to Clipboard");
```

View saa parametrikseen Controller-olion, koska Controller on kuuntelija, jota View tarvitsee omien nappuloidensa asettamiseen. Konstruktorissa luodaan JMenuBar-, JMenu- sekä JMenuItem-oliot.

```
    singleplayer.setActionCommand("STARTSINGLEPLAYER");
    multiplayerHost.setActionCommand("HOSTMULTIPLAYER");
    multiplayerJoin.setActionCommand("JOINMULTIPLAYER");
    pause.setActionCommand("PAUSE");
    highscore.setActionCommand("SHOWHIGHSCORE");
    exit.setActionCommand("EXIT");
    copyIP.setActionCommand("COPYIPTOCLIPBOARD");
    singleplayer.setFocusable(false);
    singleplayer.addActionListener(controller);
    multiplayerHost.addActionListener(controller);
    multiplayerJoin.addActionListener(controller);
    pause.addActionListener(controller);
    exit.addActionListener(controller);
    highscore.addActionListener(controller);
    copyIP.addActionListener(controller);
```

Kun komponentit on luotu, niille annetaan ActionCommand-nimitys, jolla ActionController tunnistaa komponenttien aktivoinnit. Tämän jälkeen annetaan painikkeille kuuntelija `addActionListener`-metodilla.

```
        gameMenu.add(singleplayer);
        gameMenu.add(multiplayerHost);
        gameMenu.add(multiplayerJoin);
        gameMenu.add(pause);
        gameMenu.add(highscore);
        gameMenu.addSeparator();
        gameMenu.add(exit);
        otherMenu.add(copyIP);
        jmenubar.add(gameMenu);
        jmenubar.add(otherMenu);
        singleplayer.setAccelerator(KeyStroke.getKeyStroke("F1"));
        multiplayerHost.setAccelerator(KeyStroke.getKeyStroke("F2"));
        multiplayerJoin.setAccelerator(KeyStroke.getKeyStroke("F3"));
        pause.setMnemonic('P');
        pause.setAccelerator(KeyStroke.getKeyStroke("P"));
        try{
            this.setIconImage(new ImageIcon(
            TetrisMain.class.getResource("TetrisIcon.png")).getImage());
        }catch(Exception e){
            System.err.println("Failed to download icon");
        }
        this.setLocation(400, 200);
        this.setJMenuBar(jmenubar);
        this.add(view);
        this.setResizable(true);
        this.addKeyListener(controller);
        this.setTitle("Tetris");
        this.requestFocus();
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.pack();
        this.setVisible(true);
}
```

JMenu-olioihin lisätään JMenuItem-oliot. Tämän jälkeen JMenuBar-olioon lisätään kaksi JMenu-oliota. Pikanäppäimet valikon painikkeisiin voidaan toteuttaa joko `setAccelerator`- tai `setMnemonic`-metodilla. `setMnemonic` vaatii toimiakseen yhden kirjaimen, joka on painikkeen nimessä. Nimessä kyseinen kirjain alleviivataan automaattisesti. Komponentit, mukaan lukien View-luokka, lisätään TetrisMain-luokan JFrame-kehykseen. JFrame asetetaan skaalattavaksi `setResizable`-metodilla. Yläkulman X-painikkeen sulkemistoiminto määrätään `setDefaultCloseOperation(EXIT_ON_CLOSE)`–metodilla. Lopuksi JFrame pakataan `pack`-metodilla, jota käytetään kun kaikki komponentit on lisätty. Vielä piilossa ikkuna tuodaan näkyviin `setVisible`-metodilla.

4.3 Model

Model-luokka toteuttaa Runnable-rajapinnan, joka mahdollistaa Model-luokan ilmentymän rakentamisen itsenäiseksi säikeeksi. Tämä osa käsittelee teknistä toteutusta, jolla laskelmoidaan pudottaminen, odottaminen, sivuille siirtyminen sekä näkymän ja yhteyden hallinnan ohjaus.

4.3.1 Model-olion luonti

```
public Model(View view){
    this.view=view;
    this.view.setScoreHandler(scoreHandler);
    this.space=view.getSpace()[0];
    this.nextBlocks=view.getNextBlocks();
    block[0] = new Block(space);
    block[1] = new Block(space);
    block[2] = new Block(space);
    block[3] = new Block(space);
}
```

Konstruktorissa Model-luokalle annetaan View ja annetaan Modelin luonnin yhteydessä luotu ScoreHandler-olio View-olion haltuun. Model ottaa View-oliolta space-taulukon sekä seuraavien palikoiden taulukon. Lopuksi luodaan vielä 4 ruudusta koostuva palikat, jotka on tarkoitettu space-taulun arvojen liikuttamisen toteuttamiseen.

4.3.2 Pelin aloittaminen

Pelitapahtuma ei käynnisty itsenäisesti, vaan peli käynnistetään Controller-oliossa, jolloin pelaaja on valinnut joko pikanäppäinpainalluksella(F1,F2,F3) tai hiiren ylävalikon painalluksella pelimoodin. Tämän jälkeen Controller kutsuu Model-olion `newGame`-metodia.

```
public boolean newGame(int gameMode, String IP){
    try {
        if(thread!=null)thread.stop();
        view.prepGame();
        gameStates = new GameStates();
        if(gameMode==0){
            gameStates.setActive(true);
        }
        if(gameMode==1){
            cHandler = new ConnectionHandler(view,gameMode,null,gameStates);
        }
        else if(gameMode==2){
            ConnectionHandler cHandlerTest = new ConnectionHandler(view,gameMode,IP,gameStates);
            if(!cHandlerTest.hasConnection()){
                return false;
            }
            else cHandler = cHandlerTest;
        }
        this.gameMode=gameMode;
        scoreHandler.resetScore();
        for(int x = 0; x < nextBlocks.length; x++)
        {
            nextBlocks[x] = (int)(Math.random()*7);
        }
        thread = new Thread(this);
        thread.start();
    } catch (Exception e) {e.printStackTrace();return false;}
    return true;
}
```

Uusi peli saa parametreiksi pelimoodin, sekä mahdollisen IP-osoitteen, mikäli ollaan tekemässä asiakkaan moninpeliä. gameMode-arvon ollessa 0 tarkoitetaan yksinpeliä. 1 tarkoittaa moninpeliä, jossa pelaaja toimii palvelimena. 2 tarkoittaa moninpeliä, jossa pelaaja toimii asiakkaana, joka liitetään palvelimeen. Aluksi tarkistetaan, että luokan säie ei ole päällä, jonka jälkeen tehdään alkutoimenpiteet View-luokkaan. Tämän jälkeen luodaan GameStates luokka. Mikäli kyseessä on yksinpeli, asetetaan GameStates automaattisesti positiiviseksi. Mikäli kyseessä on kaksinpeli, jossa pelaaja on päättänyt toimia palvelimena, käynnistetään ConnectionHandler, joka on vastuussa kaikesta verkkopelin liikenteestä. Mikäli `newGame`-metodi epäonnistuu, metodi palauttaa kutsujalle negatiivisen boolean arvon, ilmoittaen uuden pelin luonnin epäonnistuneen. Teoriassa tämän pitäisi vain tapahtua asiakkaan tapauksessa, jolloin liityttävä osoite ei omaa palvelinta. Jos `newGame`-metodi menee läpi, niin käynnistetään säie, jota ajetaan `run`-metodissa.

4.3.3 Pudottaminen säikeessä

Säikeessä tuotetaan aikaan pohjautuvat pudotukset sekä tehdään muutoksia pudotuksen jälkeen tapahtuvissa operaatiossa. Säikeessä tuotetaan sekä yksinpelin että muutokset ja tästä syystä tarvitaan tarkistuksia moninpelin varalle. Toisaalta tämä lisää ylimääräistä tarkistustaakkaa, mutta vähentää koodin määrää.

```
public void run(){
    int waitTime=400;
    if(gameMode==1)while(!cHandler.hasConnection());
    if(gameMode==2)if(!cHandler.hasConnection())return;
    if(gameMode!=0)cHandler.startReader();
```

Alkutoimina asetetaan säikeen odotusaika, eli toisin sanoen kappaleen putoamisnopeus. Jos toimitaan moninpelin palvelimena, odotetaan, että cHandler saa yhteyden liittyvään asiakkaaseen. Jos ollaan moninpelissä asiakkaana, ei turhaan jäädä odottamaan, mikäli palvelin ei ole vielä pystyssä. Jos tämän jälkeen on päästy pidemmälle, niin kaksinpelitapauksissa käynnistetään cHandler-luokassa lukija, jonka tarkoitus on päivittää vastustajan puolella tehdyt siirrot omalle koneelle.

```
try{
    while(gameStates.isActive()){
        takeFromList(nextBlocks[0]);
        view.repaint();
        dropping = true;
        canMove = true;
        while(dropping && gameStates.isActive()){
            Thread.sleep(waitTime);
            if(gameStates.isActive())dropDown();
            if(gameMode!=0){
                cHandler.sendSpace();
            }
        }
```

Kun alkutoimet on tehty, poimitaan palikoita jonosta niin kauan, kunnes pelin tila ei ole enää aktiivinen. Palikan ottamisen jälkeen View päivitetään, jolloin seuraava uusi palikka näkyy jonon viimeisenä. Mikäli on käytetty välilyönnin suoraa pudotusta, palautetaan canMove positiiviseksi.

Kun säie on odottanut vuoronsa, se pudottaa palikan alaspäin. Mikäli ollaan moninpelissä, niin cHandler lähettää vastapelurille siirretyn nappulan.

```
            rowsToSend = 0;
            for(int y = 4; y < 20; y++)
            {
                boolean linecheck = true;
                for(int x = 0; x < 10 ; x++)
                {
                    if(space[x][y]==0){
                        linecheck = false;break;
                    }
                }
                if(linecheck){
                    for(int yi = y; yi > 1 ; yi--)
                    {
                        for(int x = 0; x < 10 ; x++)
                        {
                            space[x][yi]=space[x][yi-1];
                        }
                    }
                    if(gameMode==0){
                        scoreHandler.addPoints();
                        if(waitTime>125)waitTime--;
                    }
                    else rowsToSend++;
                }
            }
            view.alterRowsToSend(rowsToSend, true);
            if(gameMode!=0){
                cHandler.getEnemyDrops();
            }
            for(int x = 0; x < 40; x++){
                if(space[x%10][x/10]!=0){gameStates.setActive(false);}
            }
        }
        endGame();
    }
    catch (Exception e){e.printStackTrace();}
}
```

Kun palikkaa ei voi pudottaa alaspäin se siirtyy pois pudotus tilasta, jolloin tarkistellaan putoamisen jälkeen tapahtuvia toimenpiteitä. Ensiksi etsitään kahdessa sisäisessä for-loopissa, että onko yhtään riviä, joka olisi täysi pudotetuista palikoista. Mikäli rivi löytyy, niin linecheck palauttaa arvon true, jolloin kaikki rivin yläpuoleiset rivit siirtyvät yhden alaspäin. Mikäli kyseessä on yksinpeli sekä putoamisaika on suurempi kuin 125 millisekuntia, niin ohjelma vähentää pudotusajasta yhden millisekunnin. Jos kyseessä on kaksinpeli, niin ohjelma lisää vastustajalle lähetettäviä rivejä. Tämän jälkeen tuodaan vastustajan lähettämät rivit. Tämän jälkeen tarkistetaan, onko neljällä ensimmäisellä rivillä palikoita. Mikäli näin on, niin ohjelma keskeytetään ja siirrytään `endGame`-metodiin.

4.3.4 Pelin lopettaminen

Peli lopetetaan kun säikeestä poistutaan. Pelin loputtua säikeen ei tarvitse tehdä toistuvia komentoja.

```
private void endGame() {
    if(gameMode==0){
        view.drawEndScreen(1);
        if(scoreHandler.hasBetterScore())view.showAskName();
    }
    else{
        cHandler.sendSpace();
    }
    view.repaint();
}
```

Pelin loppuessa tarkistetaan ScoreHandler-oliolta yksinpelissä, onko pisteytys parempi kuin aikaisempi. Jos pisteytys on parempi, niin ohjelma tuottaa View olion avulla ruudulle nimensyöttölomakkeen. Mikäli kyseessä on moninpeli, ohjelma lähettää viimeisen viestin vastustajalle, joka sisältää tiedon häviämisestä. Viimeisenä päivitetään tilanne ruudulle.

4.4 Block

Block on Modelin omistama luokka. Seuraavaksi käydään läpi palikan luonti sekä siirtäminen.

4.4.1 Block-olion luonti

Kun Block-luokasta tehdään ilmentymää, niin parametrina annetaan kaksiulotteinen taulukko space.

```
public Block(byte[][] space){
    this.space=space;
}
```

Tämä taulukko toimii ohjelman ruudulla näkyvänä säiliönä, johon palikat pudotellaan. Palikka luodaan vain kerran ohjelman käynnistyessä.

4.4.2 Uusi pudotus

Kun seuraavien palikoiden jonosta nostetaan seuraava, niin palikalle annetaan uudet koordinaatit sekä tyyppi. Palikan kulma aloittaa aina nollasta.

```
public void createNewBlock(int x, int y, int type){
    angleOfBlock = 0;
    X=(byte)x;
    Y=(byte)y;
    this.type=(byte)type;
    space[x][y]=(byte)type;
}
```

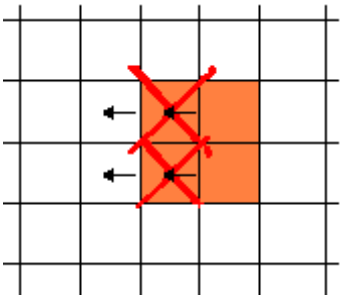
Uuden pudotuksen alkaessa, kutsutaan `createNewBlock`-metodilla. Tällä metodilla annetaan luokan sisäisiksi muuttujiksi arvot, sekä tyyppi. Arvot määrittävät sen, mihin kohtaan uutta palikkaa tullaan piirtämään. Tyyppi kertoo piirtoluokalle palikan värin sekä Modelille, miten sitä käännetään.

4.4.3 Siirron tekeminen

Palikkaa siirretään tekemällä aluksi tarkistus jokaisen palikan ruutuun.

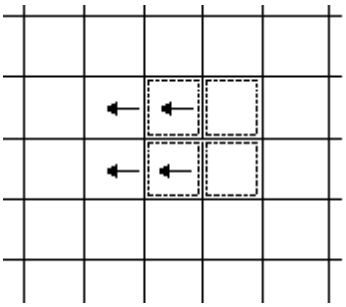
```
public void clear(){
    space[X][Y]=0;
}
```

Ensimmäisenä poistetaan ruudun säiliössä oleva arvo jokaisesta palikasta, jotta tätä seuraava tarkistus ei epäonnistu (kuva 14).



Kuva 14. Ilman palikan kadottamista, ohjelma luulisi siirron olevan mahdoton.

Kun poiston toteuttaa, niin tarkistuksessa oikeanpuoleiset ruudut saavat tietää, että siirtämiseen on tilaa (kuva 15). Toinen keino olisi tarkistaa vain ne ruudut, jotka ovat oikealla. Tämä tosin aiheuttaisi sen, että jokainen tapaus pitäisi käsitellä erikseen.



Kuva 15. Palikka tarkistelee vasemmanpuoleista tilaa.

Poistettu palikka säilyttää arvonsa Block-luokan X,Y ja type-muuttujissa. Nyt voidaan tarkistella ruutua joka sijaitsee tässä esimerkissä vasemmalla puolella.

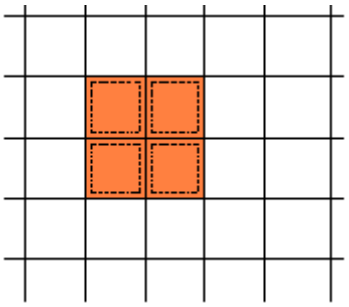
```
public boolean canIMoveLeft() {
    if(X>0)return space[X-1][Y]==0;
    return false;
}
```

`canIMoveLeft`-metodi tarkastelee aluksi, että emme ole vasemmassa reunassa, muuten taulun vasemman puoleisen arvon vertailu aiheuttaisi virheen. Jos emme ole vasemmassa reunassa, niin palautamme arvon, joka saadaan palikan ruudun vasemmanpuoleisen arvon tarkastelulla. Mikäli arvo on nolla, se tarkoittaa sitä, että tilaa löytyy. Tällöin positiivinen arvo palautuu.

Tilan löytyessä, Model-luokassa siirrytään käyttämään `moveLeft`-metodia. Koordinaatit, jolla palikan ruutu piirretään, siirtyvät yhdellä vasemmalle. Tämä siirtymä tehdään myös kaikille muille ruuduille Block-taulukossa.

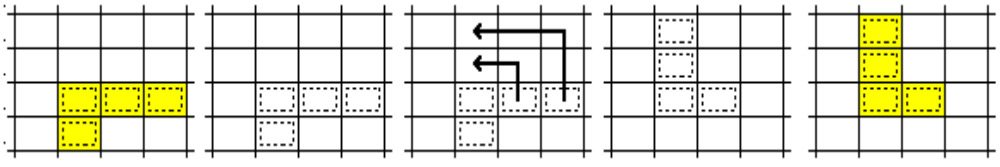
```
public void pasteBack(){
     space[X][Y]=type;
}
```

Tämän jälkeen piirretään arvot takaisin (kuva 16). Ruudulle tulostuu kaksiulotteisen taulukon arvot, ja palikan uudet koordinaatit. Palikan pudotessa pohjalle nostetaan ylös uusi palikka eikä vanhoja arvoja aseteta pois.



Kuva 16. Palikka tuodaan takaisin näkyviin.

Kääntäminen toimii samalla tavalla, mutta palikan ruutuja joudutaan tarkistamaan ja siirtämään useamman yksikön verran taulukossa ja käyttäen molempia koordinaatteja sekä perustuen myös palikan asentoon (kuva 17).



Kuva 17. L-palikan kääntäminen

4.5 View

View on luokka joka perii JPanel-luokan kirjastot. View omistaa oman perityn JPanel-olion sekä 3 muuta JPanel-muuttujaa. Näihin kuuluvat huip-

pupisteiden tulostamiseen, huippupisteiden nimen syöttämiseen sekä peliin liityttäessä tapahtuvan palvelimen IP:n syöttöön varatut kentät.

4.5.1 Piirtämiskutsu

Piirtäminen tapahtuu `paintComponent`-metodissa, jota kutsutaan `repaint`-metodilla. Parametriksi se saa Graphics-olion. Graphics-luokassa on määritelty joukko metodeita kuvan muodostamiseen ja piirtämiseen.

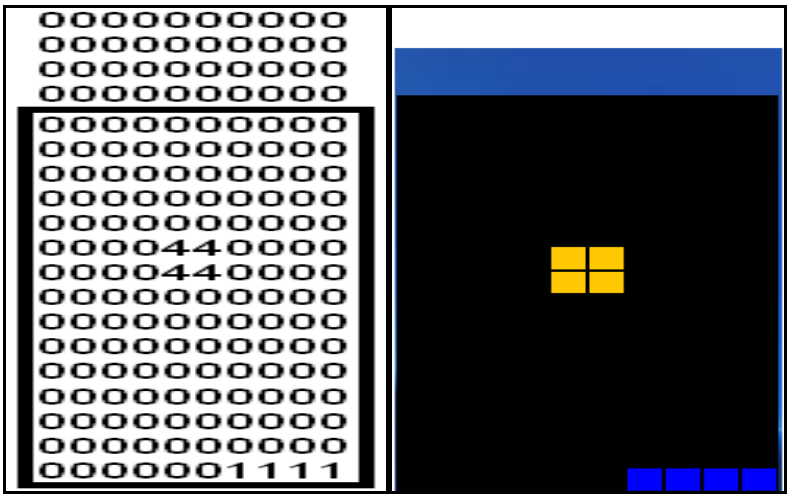
```
public void paintComponent(Graphics g)
{
    Graphics2D g2D = (Graphics2D)g.create();
    if(duelMode)g2D.scale(((double)this.getWidth())/600, ((double)this.getHeight())/375);
    else g2D.scale(((double)this.getWidth())/300, ((double)this.getHeight())/375);
    switch(endState)
    {
        case -1: g2D.drawImage(wallpapers[0], 0, 0, mapsizeX,  mapsizeY, this); break;
        case 0:  g2D.drawImage(wallpapers[1], 0, 0, mapsizeX*2,mapsizeY, this); break;
        case 1:  g2D.drawImage(wallpapers[2], 0, 0, mapsizeX,  mapsizeY, this); break;
        case 2:  g2D.drawImage(wallpapers[3], 0, 0, mapsizeX*2,mapsizeY, this); break;
        case 3:  g2D.drawImage(wallpapers[4], 0, 0, mapsizeX*2,mapsizeY, this); break;
        case 4:  g2D.drawImage(wallpapers[5], 0, 0, mapsizeX*2,mapsizeY, this); break;
    }
    if(endState==0)
    {
        drawNextBlocks(g2D);
        drawPlayer1(g2D);
        if(duelMode)drawPlayer2(g2D);
    }
    if(askHostAddressField.isVisible())
        askHostAddressField.setLocation(this.getWidth()/2-60, this.getHeight()/2-55);
    if(askNameField.isVisible())
        askNameField.setLocation(this.getWidth()/2-60, this.getHeight()/2-45);
    if(showHighScoreField.isVisible())
        showHighScoreField.setLocation(this.getWidth()/2-60, this.getHeight()/2-55);
}
```

Aluksi muunnetaan Graphics muuttuja Graphics2D muuttujaksi, joka on Graphics-olion sisään kirjoitettu edistyneempi piirtoalusta. Tämän avulla voidaan muun muassa skaalata piirtoalustaa käyttäjän tahdon mukaisesti. Yksinpelissä aloituskokona on 300x375 pixeliä ja moninpelissä 600x375 pixeliä, joten skaalaus toimii jakamalla alkuperäinen koko ja kertomalla nykyinen koko. Tämän jälkeen piirretään taustakuva sen mukaan, missä pelitilassa ollaan. Mikäli peli on yhä käynnissä, niin tulostetaan pelaajan seuraavat palikat sekä yksinpeliruutu. Jos kaksinpeli on valittuna, niin piir-

retään myös vastustajan ruudukko. Lopuksi siirretään luokan sisäiset JPanel-kehykset ruudun keskelle, mikäli ovat esillä.

4.5.2 Peliruudukon piirtäminen

Peliruudukko perustuu kaksiulotteisen byte-taulun arvojen piirtämiseen. Arvo 0 tarkoittaa tyhjää ruutua. Muut arvot sen sijaan käskevät Graphics-työkalua piirtämään värin mukaan neliön sen taulukossa merkatun sijainnin paikkaan (kuva 18).



Kuva 18. Ohjelman piirto taulukosta ruudulle.

Kahta ensimmäistä riviä ei näytetä, vaan ne on varattu sille, että moninpelissä nostettaessa ylöspäin, toiselle riville saattaa tulla päällekkäisiä nostoja.

```
private void drawPlayer1(Graphics g) {
    g.setColor(Color.black);g.fillRect(84, 40, 200, 320);
    for(int y = 2; y < 20; y++)
    {
        for(int x = 0; x < 10; x++)
        {
            if(space[0][x][y]!=0)
            {
                switch(space[0][x][y])
                {
                    case 1: g.setColor(Color.blue);          break;
                    case 2: g.setColor(Color.pink);          break;
                    case 3: g.setColor(Color.yellow);        break;
                    case 4: g.setColor(Color.orange);        break;
                    case 5: g.setColor(Color.green);         break;
                    case 6: g.setColor(Color.cyan);          break;
                    case 7: g.setColor(Color.LIGHT_GRAY);    break;
                    case 8: g.setColor(Color.WHITE);         break;
                }
                g.fillRect(20*x+85, 20*y-39, 18, 18);
                if(y<4){
                    g.setColor(Color.black);
                    g.drawRect(20*x+84, 20*y-40, 19, 19);
                }
            }
        }
    }
    if(!duelMode){
        g.setColor(Color.black);
        g.setFont(font[0]);
        g.drawString("Score", 10, 17);
        g.drawString(String.valueOf(ScoreHandler.getScore()), 10, 35);
    }
}
```

Tässä piirretään pelaajan ruudukko. Aluksi taustalle värjätään musta suorakulmio, jonka jälkeen käydään kahdessa sisäisessä for-loopissa läpi space-taulukko, josta ruudukko muodostuu. Aluksi tarkistetaan, että arvo ei ole nolla, jolloin tiedetään, että ruutuun tulee piirtää. Tämän jälkeen valitaan ruudulle oikea väri sen mukaan, mikä palikka ollaan pudottamassa tai on pudotettu. Tämän jälkeen tehdään täysi suorakulmio, jonka reunat väritetään mustalla. Yksinpelin tapauksessa piirretään ScoreHandler-olioon kerätyt pisteet.

4.5.3 Seuraavat palikat

Tämä metodi tulostaa pelaajan ruudukon vasemmalle puolelle rivin ilmestyvistä palikoista. Seuraavassa esimerkissä käydään sinisen suoran palikan piirtäminen.

```
private void drawNextBlocks(Graphics g) {
    final int moveX = 10;
    switch(nextBlocks[0])
    {
        case 0:
            g.setColor(Color.blue);
            g.fillRect(moveX, 60, 40, 10);
            g.setColor(Color.black);
            g.drawRect(moveX, 60, 10, 10);
            g.drawRect(moveX+10, 60, 10, 10);
            g.drawRect(moveX+20, 60, 10, 10);
            g.drawRect(moveX+30, 60, 10, 10);
            break;
```

Aluksi asetetaan siirtämiseen tehty moveX-muuttuja, jolla saadaan muutettua etäisyyttä piirtoalustan vasemmasta reunasta, mikäli halutaan tätä ominaisuutta vaihtaa. Tämän jälkeen piirretään palikkakohtaisesti, ruutu kerrallaan, seuraavat palikat. Ensimmäinen palikka piirretään tuplasti isompana kuin muut, erottuen näin joukosta. Muut palikat piirretään tämän jälkeen for-loopissa. Seuraavia palikoita näkyy 10 kappaletta.

4.6 Controller

Seuraavaksi käydään näppäinten painallusten tunnistaminen, sekä niistä seuraavat operaatiot sekä leikepöydän hallitseminen.

4.6.1 Controller-olion luonti

Controller-olion luonnissa annetaan viitteet muihin valmiiksi luotuihin olioihin.

```
private TetrisMain tetrisMain;
private View view;
private Model model;

public Controller(TetrisMain tetrisMain, View view, Model model)
{
    this.tetrisMain = tetrisMain;
    this.view = view;
    this.model = model;
}
```

Controller-olion luonti on yksinkertainen toimenpide, jossa asetetaan komennettavat luokat Controller-luokan tietoon. TetrisMain-oliota tullaan tarvitsemaan pakkaamisessa, Viewiä, kenttien avaamisessa sekä sulkemisessa ja Model-oliota palikoiden suunnan muuttamiseen.

4.6.2 Näppäinten kuunteleminen

Näppäinten kuuntelu tapahtuu `keyPressed`-metodissa, jonka sisällä painikkeiden tunnistaminen tapahtuu switch-case rakenteessa.

```
public void keyPressed(KeyEvent e)
{
    if(model.getGameStates()!=null && model.getGameStates().isActive())
    {
        switch(e.getKeyCode()){
            case 32: model.spaceAction(); break;
            case 37: model.leftAction();  break;
            case 38: model.upAction();    break;
            case 39: model.rightAction(); break;
            case 40: model.downAction();  break;
        }
    }
}
```

Näppäinpainalluksia kuunnellaan `keyPressed`-, `keyReleased`- sekä `keyTyped`-metodissa. `keyPressed`-metodi aktivoituu välittömästi kun näppäintä painetaan. `keyReleased` aktivoituu kun näppäimestä päästetään irti.

4.6.3 Toimintojen kuunteleminen

Controller perii ActionListener-luokan, joka pystyy kuuntelemaan toimintoja, joita näppäinten painallukset kutsuvat.

```
if(e.getActionCommand().equals("STARTSINGLEPLAYER") &&  !view.isAskingInput()
        && (model.getGameStates()==null || !model.getGameStates().isActive()
        || model.isWaitingJoin()))
{
    model.closeConnections();
    view.hideHighScore();
    if(model.newGame(0,null))
    {
        if(view.setDuelScreenMode(false))tetrisMain.pack();
    }
}
```

Tässä edellä on yksinpelin käynnistäminen. Kun näppäintä painetaan, tarkastetaan, että sen komentonimi on oikea. Tämän jälkeen tarkistetaan View-luokalta, ettei olla syöttämässä tietoa. Tämän jälkeen tarkistetaan, että GameStates on tyhjä, peli ei ole päällä tai että ollaan vielä odottamassa yhteyden ottoa palvelimena.

Kun nämä ehdot on täytetty, niin käsketään Model-luokkaa sulkemaan kaikki yhteydet, mikäli nämä ovat jääneet edellisestä pelistä päälle, piilottamaan huippupisteiden tulostuskentän mikäli se on asetettu näkyviin. Tämän jälkeen kutsutaan Model-luokan `newGame`-metodia. Mikäli metodi on onnistunut, asetetaan mahdollinen kaksinpeli ruutu yksinpeli ruudun kokoon ja pakataan TetrisMain-luokan JFrame, jotta muutokset tulisivat voimaan.

```
if(e.getActionCommand().equals("JOINMULTIPLAYER") && !view.isAskingInput()
        && (model.getGameStates()==null || (!model.getGameStates().isActive()
        || model.isWaitingJoin())))
{
    view.hideHighScore();
    view.showAskHostAddress();
}
```

Moninpelin tekeminen palvelimena käyttää samaa ratkaisua kuin aikaisempi, mutta peliin liittyminen avaa osoitteen kyselemisen.

```
if(e.getActionCommand().equals("ACCEPTADDRESS"))
{
    view.hideHighScore();
    model.closeConnections();
    view.hideAskHostAddress();
    tetrisMain.requestFocus();
    if(model.newGame(2,view.getJoinIP()))
    {
        view.setDuelScreenMode(true);
        tetrisMain.pack();
    }
}
```

Painettaessa hyväksymistä, kyseinen painike lähettää Controllerille komennon tuottaa uusi peli, jossa hankittuun IP:hen liitytään.

```
if(e.getActionCommand().equals("COPYIPTOCLIPBOARD"))
{
    try{
        Toolkit.getDefaultToolkit().getSystemClipboard().setContents(
                new StringSelection(
                        java.net.InetAddress.getLocalHost().getHostAddress() ), this);
    }catch(Exception e1){}
}
```

Mikäli asiakkaana liittyvä pelaaja ei tiedä kaverinsa IP-osoitetta, voi kaveri lähettää osoitteen keskusteluohjelman viestillä. Metodi käynnistyy Other valikosta löytyvältä ”Copy IP to Clipboard” painikkeelta. Koneen IP-osoite tallentuu koneen leikepöydälle ja on tulostettavissa esimerkiksi näppäinyhdistelmällä CTRL+V.

4.7 ConnectionHandler

ConnectionHandler on yhteydenpitoluokka. Seuraavaksi käydään läpi yhteyden avaaminen, ylläpitäminen ja sulkeminen, sekä tietojen lähettäminen sekä vastaanottaminen.

4.7.1 ConnectionHandler-olion luonti

ConnectionHandler luodaan, kun pelaaja on valinnut moninpelin.

```
public class ConnectionHandler {
    private ServerSocket serverSocket;
    private Socket socket;
    private OutputStream outputStream;
    private InputStream inputStream;
    private View view;
    private byte[][][] space;
    private GameStates gameStates;
    private final int PORT = 4444;
    private boolean waitingJoin;

    public ConnectionHandler(View view, int gameMode, String IP, GameStates gameStates){
        try
        {
            if(gameMode==1)
            {
                new hostThread();
            }
            else
            {
                socket = new Socket(IP,PORT);
                outputStream = socket.getOutputStream();
                inputStream = socket.getInputStream();
            }
            this.view=view;
            this.space=view.getSpace();
            this.gameStates=gameStates;
            gameStates.setActive(true);
        }
        catch (Exception e) {}
    }
```

ConnectionHandler-luokan luonnin yhteydessä tarkistetaan käyttäjän valitseman pelitilan perusteella joko palvelimen tai asiakkaan rooli. Mikäli kyseessä on palvelimen rooli, käynnistetään luokan sisäinen aliluokka, jonka tarkoitus on toimia palvelimen odottavana säikeenä. Mikäli ollaan yhdistämässä, niin mitään säikeitä ei tarvitse tehdä, vaan tuotetaan pistoke, joka liitetään parametreista saatuun IP-osoitteeseen. Mikäli osoite on tyhjä, niin pistoke yrittää yhdistää oman koneen IP:hen. Yhdistämisen jälkeen asiakas luo OutputStream- sekä InputStream-tietovirran, joilla asiakas keskustelee palvelimen kanssa. Tämän jälkeen linkitetään parametreistä saadut arvot ja aktivoidaan peli.

4.7.2 Palvelinsäie

hostThread on palvelinsäie-luokka, jota kutsutaan, kun halutaan siirtyä uuteen peliin palvelimena.

```
private class hostThread implements Runnable{
    public hostThread()
    {
        Thread t = new Thread(this);
        t.start();
    }
    public void run()
    {
        waitingJoin=true;
        try
        {
            serverSocket = new ServerSocket(PORT);
            try
            {
                socket = serverSocket.accept();
            }
            catch (SocketTimeoutException e) {}
            outputStream = socket.getOutputStream();
            inputStream = socket.getInputStream();
            waitingJoin=false;
        } catch(Exception e){}
    }
}
```

Konstruktorissa säie käynnistetään. Aliluokka perii luonnollisesti kaikki ylemmän luokan arvot ja saa näin käyttää serverSocket muuttujaa. ServerSocket on olio, jolla odottava yhteys avataan. Kun serverSocket käyttää `accept`-metodia, se jää pysyvästi odottamaan kunnes joko serverSocket lopetetaan tai se on odottanut määrätyn ajan. Mikäli joku yhdistää onnistuneesti, palvelin luo OutputStream- sekä InputStream-tietovirran, joilla keskustelu hoituu asiakkaan kanssa.

4.7.3 Tietojen paketointi

Socket on siitä erinomainen, että se mahdollistaa tiedon lähettämisen ilman, että mitään ylimääräistä tarvitsee syöttää. Kaikki tieto paketin perille menosta on valmiina Socket-luokan sisällä. Ainoa syöte on toiselle koneelle haluttava tieto. Tietojen lähettäminen tapahtuu byte-taulukkojen avulla, joilla on omat rajoitteensa.

Byte tarkoittaa tavua eli kahdeksasta bitistä muodostuvaa binäärilukua. Jokainen bitti omaa kaksi mahdollista arvoa, 0 ja 1. Kun näitä bittejä on 8 peräkkäin, niin biteillä on 256 mahdollista järjestystä. Tavun suurin arvo

on 127 ja pienin -128, sillä ensimmäinen bitti määrittää, onko luku negatiivinen.

4.7.4 Tiedon lähettäminen

Tiedot lähetetään `sendSpace`-metodilla, jota kutsutaan jokaisella muutoksella omassa space-taulukossa.

```
public synchronized void sendSpace(){
    try
    {
        byte[] send = new byte[202];
        for(int i=0; i < 200; i++){
            send[i]=space[0][i%10][i/10];
        }
        int s = view.getRowsToSend();
        send[200]=(byte)(s/2);
        view.alterRowsToSend((byte)((s/2)*2),false);
        if(!gameStates.isActive()){
            send[201]=1;
            view.drawEndScreen(3);
        }
        outputStream.write(send);
    }catch(Exception e){gameStates.setActive(false);view.drawEndScreen(4);}
}
```

Kun `sendSpace`-metodia kutsutaan, tuotetaan lähetettävä taulu, joka sisältää pelaajan ruudukon sisällön, sekä yhden arvon lähetettäville riveille, jotka vastustaja saa omista romautetuista riveistä. Kun saatuja rivejä on kaksi kappaletta, eli ollaan romautettu kaksi riviä, niin vastustajalle tulee yksi rivi lisää. Toinen arvo on varattu tilanteen selvittämiselle. Mikäli peli hävitään, lähetetään vastustajalle tieto siitä. Mikäli tapahtuu virhe, niin silloin lähetys on epäonnistunut. Vika tähän on joko vastustajan tekemä ohjelman sulkeminen tai muu yhteysongelma. Silloin tuotetaan ruudulle pelitila 4, jossa ilmoitetaan pelaajalle yhteyden katkenneen.

4.7.5 Tiedon vastaanottaminen

Model-luokkaa käsitellessä mainittiin `startReader`-metodi. Tämän tarkoitus on päivittää ruudulle vastustajan siirrot.

```
public void startReader(){
    new readerThread();
}
```

Tällä komennolla käynnistetään säie, jonka tarkoitus on vastaanottaa vastustajan lähettämiä paketteja while-loopissa.

```
private class readerThread implements Runnable{
    public readerThread()
    {
        Thread t = new Thread(this);
        t.start();
    }
    public void run()
    {
        while(gameStates.isActive())
        {
            readSpace();
            view.repaint();
        }
    }
}
```

Kun peli on käynnissä, lukijasäie pysyy while-loopin sisällä lukien vastustajan ruudukon uudet tilat sekä päivittäen siirrot ohjelman omaan ruudukkoon.

```
public void readSpace(){
    if(gameStates.isActive())
    {
        try{
            byte[] receive = new byte[202];
            inputStream.read(receive);
            for(int i=0; i < 200; i++){
                space[1][i%10][i/10]=receive[i];
            }
            view.alterRowsToReceive(receive[200], true);
            if(receive[201]==1)
            {
                gameStates.setActive(false);
                view.drawEndScreen(2);
            }
        }catch(Exception e){gameStates.setActive(false);view.drawEndScreen(4);}
    }
}
```

Tiedot vastaanotetaan `readSpace`-metodissa, jossa inputStream-muuttujan `read`-metodilla luetaan arvot taulukkoon.

Arvot 0-199 kopioidaan vastustajan ruudukkoon, jonka jälkeen vastaanotetaan vastustajan pudottamat rivit, jotka siirtyvät omaan ruudukkoon. Tämän jälkeen tarkistetaan, onko pelaaja hävinnyt.

4.8 ScoreHandler

Pisteiden käsittely tapahtuu ScoreHandler luokassa, joka pitää tietoa aktiivisen pelin pisteistä sekä pelin aikana saaduista huippupisteistä.

4.8.1 ScoreHandler-olion luonti

Kun ScoreHandler luodaan, niin konstruktorissa luetaan huippupisteet tiedostosta, joka sijaitsee samassa polussa kuin ohjelma. Mikäli tiedosto puuttuu tai lukemisessa tapahtuu virhe, niin silloin tulee poikkeus ja arvot säilyvät tyhjinä.

```
private static final long serialVersionUID = 1L;
private int score;
private int highScore;
private String name;
public ScoreHandler(){
    try
    {
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("Highscore.dat"));
        ScoreHandler hs = (ScoreHandler)in.readObject();
        this.setName(hs.getName());
        this.setHighScore(hs.getHighScore());
        in.close();
    }
    catch(Exception e){}
}
```

Aluksi avataan FileInputStream, jolla tiedosto tuotetaan luettavaan muotoon. Tämän jälkeen avataan ObjectInputStream, jolla saadaan muunnettua tiedosto objektiin. Kun tiedosto on objektissa, se uudelleennimetään ScoreHandler-olioksi. Tämä vaatii toimiakseen Serializable-rajapinnan.

4.8.2 Tiedostoon tallentaminen

Tallentaminen tapahtuu, kun pisteytys on suurempi kuin nykyinen huippupisteytys.

```
public void saveFile(){
    try {
        ObjectOutputStream out = new ObjectOutputStream(
            new FileOutputStream("Highscore.dat"));
        highScore=score;
        out.writeObject(this);
        out.close();
    } catch (Exception e) {System.err.println("Error in saving file");}
}
```

Tiedoston tallentaminen tapahtuu ScoreHandler luokan `saveFile`-metodilla. HighScore luo FileOutputStream luokan, jolle se määrää syötettävän tiedoston nimeksi "Highscore.dat". ObjectOutputStream ilmentymällä FileOutputStream siirtää ohjelman datan tiedostoon.

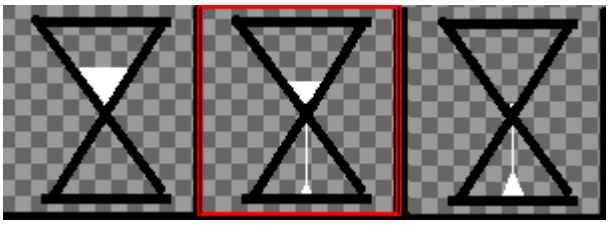
4.9 Ongelmatilanteet

Tässä käydään lyhyesti osa ohjelman toteuttamisvaiheessa tulleista ongelma kohdista ja niiden ratkaisuista.

4.9.1 Kuvien avaaminen

Kuvia ei voi piirtää suoraan ruudulle Viewin Graphics-metodilla vain yksinkertaisesti kutsumalla niitä Graphics-metodeissa, vaan kuvat täytyy ennalta ladata joko Image tai BufferedImage muuttujaan.

Imagea käytetään yleisesti vain tiedostojen siirtämiseen. Kuvien avaaminen tapahtuu tässä toteutuksessa BufferedImage-muuttujan avulla. BufferedImage mahdollistaa esimerkiksi animaatioiden tekemisen `getSubImage`-metodilla (kuva 19). Sillä voidaan palauttaa kuvasta osa. Näin muuttamalla piirtoaluetta, voidaan toteuttaa esimerkiksi kuvan animaatio.



Kuva 19. BufferedImage voi hakea kuvan osan.

4.9.2 Kuvien tallentaminen

Olin ohjelman teon aikaisessa vaiheessa tehnyt ikonit ja taustakuvat suoraan JAR-tiedoston viereen. Myöhemmin siirsin ne kansioon INC ja avauspolun joutui vaihtamaan sen mukana. Olin pitkään tiennyt, että kuvia voi sisällyttää JAR-tiedostoihin, mutta en tiennyt tarkalleen miten. Minulle selvisi, että itseasiassa kuvat kyllä menevät ohjelman koodin mukana tiedostoon, mutta niitä tulee etsiä eri tavalla. Kuvat avataan `ImageIO.read(getClass.getResource(""))`-metodilla, jossa sisimpänä parametrinä annetaan kuvan nimi.

4.9.3 Näppäinkuuntelijan toimimattomuus

Kun hiirellä valitaan komponentti, niin joissain tapauksissa kohdistus, joka aikaisemmin oli voimassa, menetetään ja klikattu komponentti saa kohdistuksen. Ongelmia alkoi muodostua, kun painikkeita ja kenttiä alettiin lisäämään ohjelmaan. Pian huomasin, että esimerkiksi JButtonia painettuani ohjelma lakkasi kuuntelemasta näppäinpainalluksia.

Ongelma oli siinä, että kyseiset J-komponentit siirsivät ohjelman kohdistuksen itseensä, ja näin kadottivat kohdistuksen JFramesta, johon se alunperin kuului ja jonka painalluksia KeyListener oli laitettu kuuntelemaan. Pikainen ratkaisuni oli lisätä jokaiseen painikkeeseen `setFocusable`-metodi, jossa asetettiin parametrin arvoksi negatiivinen, jolla poistettiin kohdistuksen ottaminen käytöstä. Ongelma oli kuitenkin yhä voimassa, sillä nimensyöttö huipputuloksissa vaati edelleen väliaikaista kohdistusta, eikä tätä kohdistusta voinut poistaa, koska nimi kenttään täytyi syöttää nimi.

```
if(e.getActionCommand().equals("ACCEPTNAME"))
{
    if(view.hideAskName())tetrisMain.requestFocus();
}
```

Ratkaisuksi löytyi `requestFocus`-metodi, jolla kohdistus voidaan palauttaa nimen hyväksyntä-napin painalluksen jälkeen alkuperäiselle omistajalle.

4.9.4 Asiakkaiden odottaminen

Palvelinohjelma tyypillisesti odottaa niin kauan kunnes sen porttiin liitytään. Tämä odottaminen aiheuttaa koko ohjelman jähmettymisen, koska ohjelman toimintoja suorittava säie jää odottamaan liittymistä. Ongelma ratkaistaan ulkoistamalla odottaminen säikeelle, jolloin muut toiminnot odottamisen lisäksi ovat voimassa.

4.9.5 Yhteyden kaatuminen

Yhteys voi kaatua monella tavalla. Vastapelaaja voi sulkea ohjelman. Yhteys voi mennä poikki, sekä ohjelma voi kaatua kriittiseen virheeseen. Tällaista tapahtumaa ei voida estää, mutta tapahtumasta voidaan selvitä. Socket kutsuu tyypillisesti yhteyden kaatuessa ”SocketException”-virhettä, joka voidaan kaapata try-catch menetelmällä. Kun yhteys kaatuu, voidaan tila raportoida ohjelmalle kaatamatta sitä.

Tämän mahdollisuus kannattaa huomioida jo ohjelman teon alkuvaiheissa, koska muuten kriittisen yhteyden kaatumisen seurauksena voidaan kaataa koko palvelin ja samalla muut siinä olevat asiakkaat.

4.9.6 Yhteyden muodostaminen

Useimmissa järjestelmissä on palomuuri, jota ohjelma ei voi ohittaa. Palomuuri estää kaiken sallimattoman sisään tulevan liikenteen. Ratkaisuna on toteuttaa poikkeus palomuuriin, jossa yhteyden muodostus sallitaan käyttöjärjestelmästä riippuen Java- tai Javaw-tiedostossa.

5 YHTEENVETO

Tässä työssä käytiin läpi Javalla toteutetun Tetrikseen pohjautuvan verkkopelin toteutusta sekä kuvattiin ohjelman rakennetta, että yksityiskohtaista arkkitehtuuria sekä annettiin opastusta Graphics2D-piirtokirjaston ja Swing-komponenttien toimintaan. Työssä opetettiin käyttämään ActionListener- ja KeyListener-kuuntelijoita. Aluksi käytiin läpi työn aloittamiseen vaikuttanutta historiaa, jonka jälkeen perehdyttiin toteutettuun ohjelman termistöön. Tämän jälkeen esitettiin ohjelma yksityiskohtaisesti sekä ratkaistiin ongelmatilanteita, joita ohjelman tekemisen aikana oli tullut ilmi.

Kuten edellä voidaan huomata, on Java-ohjelmistokielessä helpot ja hyvät välineet yhteyden muodostamiseen sekä syötteiden kuuntelemiseen. Ja-

valla voidaan helposti luoda valikoita ja piirtää ruudulle vaivattomasti monimutkaisempiakin muotoja. Javan ominaisuuksien vuoksi se on hyvä ohjelmistokieli niin aloitteleville ohjelmoijille kuin ammattilaisille.

Toivon työni antavan hyvän pohjan pelien sekä muiden graafisten ohjelmistojen tekemiseen.

Lähdeluettelo

[1.] Classic Gaming: Pong ClassicGaming.com's Museum. [online-tietokanta] Saatavissa: www.classicgaming.com. Viitattu 14.11.2010.

[2.] Defense / Advanced Research Project Agency. [online-tietokanta] Saatavissa: http://www.livinginternet.com/i/ii_darpa.htm. Viitattu 16.11.2010.

[3.] Kuva saatavissa: http://thegamereviews.com/userfiles/image/Bargain%20Bin/01-21-09/Doom.jpg. Viitattu 12.11.2010.

[4.] RPG Vault [online-tietokanta] Saatavissa: http://rpgvault.ign.com/articles/837/837389p1.html. Viitattu 17.11.2010.

[5.] Valor TV [online-tietokanta] Saatavissa: http://vator.tv/news/2010-10-08-world-of-warcraft-hits-12-million-paid-users. Viitattu 17.11.2010.

[6.] Kuva saatavissa: http://www.hiddenpugmarks.com/wp-content/NefarianRaidBlackwing.jpg. Viitattu 16.11.2010.

[7.] Atari HQ [online-tietokanta] Saatavissa: atarihq.com http://www.atarihq.com/tsr/special/tetrishist.html. Viitattu 11.11.2010.

[8.] Kuva saatavissa: http://pici.se/pictures/FiwMAsVVC.png. Viitattu 16.11.2010.

[9.] Kuva saatavissa: http://upload.wikimedia.org/wikipedia/fi/6/66/Coloris.gif. Viitattu 18.11.2010.

[10.] Kuva saatavissa: http://allaboutsymbian.com/software/reviews/images/sdm_s60_02.jpg. Viitattu 18.11.2010.

[11.] Kuva saatavissa: http://images5.gry-online.pl/galeria/galeria_duze2/683999312.jpg. Viitattu 18.11.2010.

[12.] Kuva saatavissa: http://www.gameclassification.com/files/games/Rampage-Puzzle-Attack.png. Viitattu 18.11.2010.

[13.] Kuva saatavissa: http://blog.cfelde.com/wp-content/uploads/2010/06/cpp_vs_java_diagram.png. Viitattu 18.11.2010.